**Міністерство Освіти І НАУКИ України**

**Національний університет "Львівська політехніка"**

ІКНІ

Кафедра **ПЗ**

### ЗВІТ

До лабораторної роботи № 5

**З дисципліни:** *“Алгоритми та структури даних”*

**На тему:** *“Метод швидкого сортування”*

**Лектор:**

доц. каф. ПЗ

Коротєєва Т.О.

**Виконав:**

ст. гр. ПЗ-26

Матвіїв Т.Т.

**Прийняв:**

асист. каф. ПЗ

Цимбалюк Т.М.

« \_\_\_\_ » \_\_\_\_\_\_\_\_ 2020 р.

∑= \_\_\_\_\_ .

Львів – 2020

**Тема роботи:**  Метод швидкого сортування.

**Мета роботи:** Вивчити алгоритм швидкого сортування. Здійснити програмну реалізацію алгоритму швидкого сортування. Дослідити швидкодію алгоритму швидкого сортування.

**TЕОРЕТИЧНІ ВІДОМОСТІ**

Швидке сортування (англійською «Quick Sort») — алгоритм сортування, добре відомий, як алгоритм розроблений Чарльзом Хоаром, який не потребує додаткової пам’яті і виконує у середньому O(n∙log(n)) операцій, однак, у найгіршому випадку робить O() порівнянь. Оскільки алгоритм використовує дуже прості цикли і операції, він працює швидше інших алгоритмів, що мають таку ж асимптотичну оцінку складності.

В основі алгоритму лежить принцип «розділяй та володарюй» (англійською «Divide and Conquer»). Ідея алгоритму полягає в переставлянні елементів масиву таким чином, щоб його можна було розділити на дві частини і кожний елемент з першої частини був не більший за будь-який елемент з другої. Впорядкування кожної з частин вудбувається рекурсивно. Алгоритм швидкого сортування може бути реалізований як на масиві, так і на двобічнму списку.

Швидке сортування є алгоритмом на основі порівнянь, і не є стабільним.

Час роботи алгоритму сортування залежить від збалансованості, що характеризує розбиття. Збалансованість, у свою чергу залежить від того, який елемент обрано як опорний (відносно якого елемента виконується розбиття). Якщо розбиття збалансоване, то асимптотично алгоритм працює так само швидко як і алгоритм сортування злиттям. У найгіршому випадку, асимптотична поведінкаалгоритму настільки ж погана, як і в алгоритму сортування включенням.

* Найгірше розбиття. Найгірша поведінка має місце у тому випадку, коли процедура, що виконує розбиття, породжує одну підзадачу з (n – 1) елементом, а другу — з 0 елементами. Нехай таке незбалансоване розбиття виникає при кожному рекурсивному виклику. Для самого розбиття потрібен час Θ(n). Тоді рекурентне співвідношення для часу роботи, можна записати наступним чином: T(n) = T(n – 1) + T(0) + Θ(n) = T(n – 1) + Θ(n). Розв’язком такого співвідношення є: T(n) =Θ().
* Найкраще розбиття. В найкращому випадку процедура поділу ділить задачу на дві підзадачі, розмір кожної з яких не перевищує (). Час роботи, описується нерівністю: T(n) ≤ + Θ(n). Тоді: T(n) = O(n∙log(n)) — асимптотично найкращий час.
* Середній випадок. Математичне очікування часу роботи алгоритму на всіх можливих вхідних масивах є O(n∙log(n)), тобто середній випадок ближчий до найкращого.

В середньому алгоритм працює дуже швидко, але на практиці, не всі можливі вхідні масиви мають однакову імовірність. Тоді, шляхом додання рандомізації вдається отримати середній час роботи в будь-якому випадку. В рандомізованному алгоритмі, при кожному розбитті в якості опорного обирається випадковий елемент.

**ЗАВДАННЯ**

1. Ознайомитись з теоретичним матеріалом.
2. В обраному середовищі програмування створити віконний проект та написати програму, яка реалізує алгоритм швидкого сортування згідно варіанту.
3. Передбачити вивід у вікні вхідних даних, результатів виконання варіанту, проміжних результатів згідно роботи алгоритму та результуючий відсортований масив даних.
4. Оформити звіт про виконання лабораторної роботи.

**Індивідуальне завдання**

Задано двовимірний масив дійсних чисел. Замінити мінімальні елементи кожного рядка на . Впорядкувати (переставити) стовпці масиву в порядку зростання їх перших елементів.

**ХІД ВИКОНАННЯ**

Покроковий опис роботи алгоритму швидкого сортування:

**Алгоритм Q.**

Алгоритм сортування одновимірного масиву в порядку зростання.

Задано масив елементів R, n – розмір масиву R, i та j – індекси проходження по масиву, – вибраний елемент на даному кроці.

Q1. Встановлення i на початок масиву, j – на кінець.

Q2. Зафіксовуємо перший – перший елемент масиву і поки i < j

повторюємо кроки Q3, Q4, Q5.

Q3. Шукаємо перший елемент , щоб .

Q4. Шукаємо перший елемент , щоб .

Q5. Якщо i < j, міняємо і місцями, збільшуємо і на одиницю, зменшуємо j на одиницю.

Q6. Якщо і не збігається з кінцем масиву, викликаємо рекурсивно цей алгоритм зі стартом в точці і.

Q7. Якщо j не збігається з початком масиву, викликаємо рекурсивно цей алгоритм з кінцем в точці j.

Q8. Кінець.

Код алгоритму швидкого сортування і фільтрування вхідного масиву.

int **partition** (double \*\*arr, int low, int high, int size, int size2)

{

double pivot = arr[0][high]; *//* *pivot*

int i = (low - 1); *//* *Index* *of* *smaller* *element*

*for*(int i = 0; i < size; ++i)

{

output += "|";

*for*(int j = 0; j < size2; ++j)

{

output += QString::number(arr[i][j], 'g',2)+"\t";

}

output += "|";

output += "\n";

}

output += "\n";

*for* (int j = low; j <= high - 1; j++)

{

*//* *If* *current* *element* *is* *smaller* *than* *the* *pivot*

*if* (arr[0][j] < pivot)

{

i++; *//* *increment* *index* *of* *smaller* *element*

swap(&arr[0][i], &arr[0][j]);

*for*(int k = 1; k < size; ++k)

{

double temp2 = arr[k][i];

arr[k][i] = arr[k][j];

arr[k][j] = temp2;

}

}

}

swap(&arr[0][i + 1], &arr[0][high]);

*for*(int k = 1; k < size; ++k)

{

double temp2 = arr[k][i+1];

arr[k][i+1] = arr[k][high];

arr[k][high] = temp2;

}

*return* (i + 1);

}

**РЕЗУЛЬТАТИ**

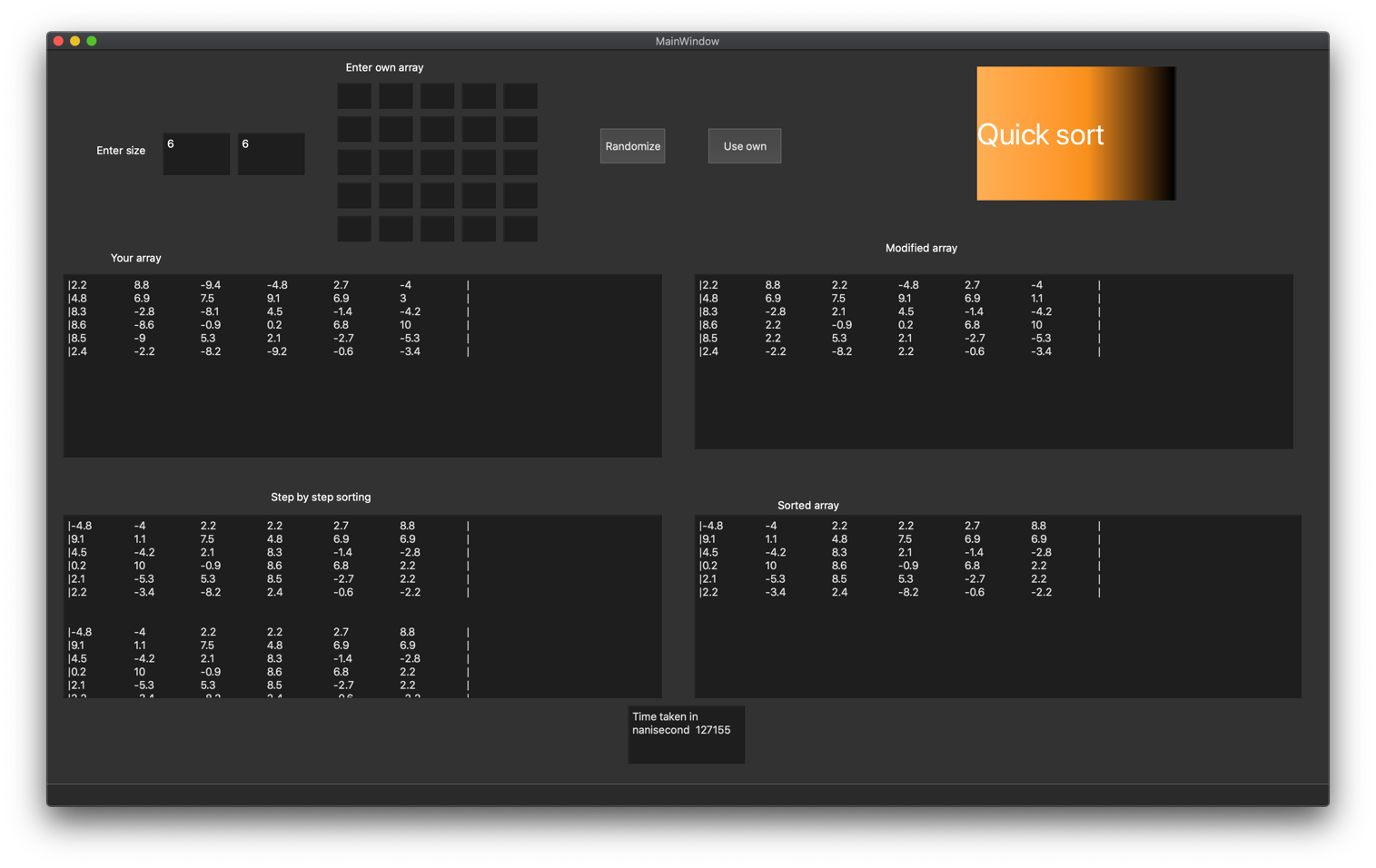


Рис. 1. Результати роботи програми

**ВИСНОВКИ**

На даній лабораторній роботі я ознайомився з алгоритмом швидкого сортування. Основна ідея методу полягає в тому, що ми порівнюємо крайні елементи масиву до базового елементу рухаючись до середини з обох кінців і міняємо їх місцями за потреби. Після того базовий елемент розбиває масив на два підмасиви, до яких застосовують цей алгоритм у рекурсії. Складність даного алгоритму в середньому є O(n∙log(n)), де n – кількість елементів у масиві.

5 16 13 8 6 1 2

5 **2** 13 8 6 1 **16**

5 2 **1** 8 6 **13** 16

5 2 1 **6 | 8** 13 16 r>l

5 2 1 6 8 13 | 16

**1** 2 | **5** 6 8 13 16

1 2 5 6

1 2 5 6 8 13 16